ECS7016P - Interactive Agents and Procedural Generation

Coursework: Unity Project (Undersea Explorers)

Adarsh Gupta - 220570653

URL for video documentation: https://youtu.be/UdHYnl_n6os

Version of Unity used: 2022.1.19f1

Generator (Level_Generator_Script.cs):

The generator works by generating a random level with a specified fill percentage. The function "createLevel()" is used to achieve this. It requires the following three parameters: width, height, and fill percentage of the level. A 2D array for the level is first created, and the level's width and height combined account for its size. The level's tile is represented by the respective integer value in the array. Water (no tile) is represented by a value of "0", while a cave tile is represented by a value of "1". The possibility that a tile will be a water or a cave tile is decided by the fill percentage variable after the "createLevel()" function has filled the array with random values.

After that, the "cellularAutomata()" function smooths the level. It takes five parameters: the level itself, the number of iterations (number of times the cellular automata algorithm should be applied), the cellular automata number (number that specifies how many neighbouring tiles should be filled in order to fill a given tile), and the width and height of the level. It then calls another function to return the neighbouring number of tiles to apply the rules of cellular automata.

The number of neighbouring tiles that are centred on the current tile is then returned by the "getNeighbourTilesCount()" function. It requires five parameters: the level, the current tile's x and y coordinates, as well as the level's width and height. The level is populated with tiles that follow the cellular automata's rules once they have been implemented. Then, cave tiles are created using this level of cellular automata implementation.

After all of the above is done, it is checked if the tile in the level is filled (its value is '1'). Then, the cave tiles are created by using the "Instantiate()" function. It takes three parameters: the cave tile prefab, the position, and the rotation. It creates (clones) the prefab object at the specified location and rotation. The cave tiles are then placed in the game world.

At the start of the scene, the "start()" function generates a level. Now, users can generate multiple levels by clicking the 'left mouse button'.

Agents:

With the help of the NPBehave and Unity movement AI libraries, the agents in the level can show a wide variety of different behaviours. The "UpdatePerception()" method, which returns the distance from other game objects, is used to update the agents' perception first. They then decide which behaviour is most appropriate for their situation. For each agent, the behaviours are defined separately:

Shark (Shark_Behaviour_Tree_Script.cs):

Seek(): When the shark is close to the mine or diver with respect to the distance provided, it then seeks them.

diverDestroy(): If the shark is too close to the diver, it attacks it, and the diver is teleported to a different location, depicting that the diver was killed.

mineDestroy(): If the shark is too close to the explosive mine, then the mine explodes and teleports both the shark and itself to a new location.

Wander(): If the shark is lost, i.e., it is nowhere close to a diver or mine, then it wanders randomly in the level until it is close to any of its interesting game objects.

Mermaid (Mermaid_Behaviour_Tree_Script.cs):

Flee(): When the diver is close to the mermaid, the mermaid tries to flee the diver.

diverDestroy(): If the diver is too close to the mermaid, then the diver kills the mermaid and teleports to a different location depicting that the mermaid was killed.

Seek():When the mermaid is close to a treasure or mine. Then the mermaid seeks the nearest treasure, or mine, with respect to the distance.

mineDestroy(): If the mermaid is too close to the explosive mine, the mine explodes and teleports both the mermaid and itself to a different location, depicting the mine killing the mermaid.

treasureDestroy():If the mermaid is too close to the treasure, it teleports to a different location, depicting the mermaid hiding the treasure.

Wander(): If the mermaid is lost, i.e., it is nowhere close to a treasure, mine, or diver, then it wanders randomly in the level until it is close to any of its interesting game objects.

Diver (Diver_Behaviour_Tree_Script.cs):

Flee(): When the shark, or mine, is close by, the diver tries to flee.

Seek(): When the diver is close to a treasure or mermaid. Then the diver seeks the nearest treasure, or mermaid, with respect to the distance.

treasureDestroy(): If the diver is too close to the treasure, the treasure teleports to a different location, indicating that the diver has looted the treasure.

mineDestroy(): If the diver is too close to the explosive mine, the mine explodes and teleports both the diver and itself to a different location, depicting the mine killing the diver.

Wander(): If the diver is lost, i.e., it is nowhere close to treasure, mine, shark, or mermaid, then it wanders randomly in the level until it is close to any of its interesting game objects.

Storyline:

The simulation was designed keeping in mind the real-life scenario. The shark just wants to survive (eat) and, parallel to that, wants to protect the mermaid from the diver killing her. So, the main priority of the shark is to kill the diver, which accomplishes both points. Now, since the shark is not so smart, when it sees an unknown little thing (an explosive mine), it goes to investigate what it is and gets killed by the mine.

The mermaid, on the other hand, is afraid of the diver killing her, so she always tries to flee from him. Next, if she is safe from the diver, she also wants to safeguard the treasure by hiding it in a different location, so she does that to protect the treasure from the diver. At last, this unknown object (an explosive mine) is new for her as well, so she goes to find out what it is and ends up getting killed.

As for this diver, he always tries to be safe from the shark eating him, so always flee the shark. Now, his average human psychology is greed, and he wants to kill the mermaid to make profit out of it or try to find and loot the treasure. Also, the diver is the one who planted the explosive mine to kill both the shark and the mermaid for his own safety and profit. Moreover, there can be a possibility that the diver is fleeing the shark and, in panic, gets too close to the explosive mine and gets killed by his own doing.

References:

1) Procedural Generation with Cellular Automata, Bronson Zgeb. Available at: https://bronsonzgeb.com/index.php/2022/01/30/procedural-generation-with-cellular-automata/

2) Procedural cave generation, Sebastian Lague. YouTube. Available at: https://www.youtube.com/watch?v=v7yyZZjF1z4&amp

3) Cellular automata method for generating random cave-like levels, RogueBasin. Available at: http://www.roguebasin.com/index.php/Cellular_Automata_Method_for_Generating_Random_Cave-Like_Levels

4) Constructive generation methods for dungeons and levels, Noor Shaker, Antonios Liapis, Julian Togelius, Ricardo Lopes, and Rafael Bidarra. Available at: https://qmplus.qmul.ac.uk/mod/resource/view.php?id=2201434

5) Cellular automata | procedural generation | game development tutorial, White Box Dev. YouTube. Available at: https://www.youtube.com/watch?v=slTEz6555Ts

6) Meniku/NPBehave: Event Driven Behavior Trees for Unity 3D, GitHub. Available at: https://github.com/meniku/NPBehave

7) Sturdyspoon/Unity-movement-ai: A unity library for common movement AI, GitHub. Available at: https://github.com/sturdyspoon/unity-movement-ai

8) Scripts of Lab 1: Behaviour Trees and Lab 5: Steering Behaviours.